# Finite difference simulation of 2D waves

**INF5620**

2014

## 1 General information

- Deadline: Oct 13
- Each student delivers a set of files, including a project report, in her/his GitHub repo, but we encourage collaboration.
- Each project will be assessed by a group of three students.
- Make a directory `wave_project` in the top directory of your INF5620 repo on GitHub to hold all your files of this project. Make suitable subdirectories. Include a `README` file with a short overview of the different files.
- Check that you have a `.gitignore` file, either in your home directory or in the root directory of your repo, where you list all redundant files and all (big) files that can be regenerated and that are not necessary for peer review of your project.
- Write a short report summarizing the main results. LaTeX is probably the preferred format, but there are several other options[1] too. Regardless of format, the report must be in an easy-to-read format like PDF or HTML.
- Note that the last part of this compulsory project allows you to develop the project in different directions, including visualization, high-performance computing, more advanced numerics, etc.

### 1.1 Check list for peer review

The project will undergo review by your peers, i.e., other students in the course. More precisely, a group of three students will assess three individual projects and write a short report for each project. You may use our general checklist[2] as starting point for the assessment.

[1] `http://hplgit.github.io/teamods/writing_reports/index.html`
[2] `http://tinyurl.com/opdfafk/pub/web-INF5620/exercise_checklist.html`

Name the feedback file `FEEDBACK_NOT_PASSED.txt` if the group's decision is that the project is not passed, otherwise name the file `FEEDBACK.txt`. The teachers will follow up on projects that are considered not passed, and the new version will be assessed by a teacher.

### 1.2 Background material

**Main document.**

The various building blocks needed in this project is found in the text *Finite difference methods for wave motion* in the course notes[a]. Observe that there are also links to a compact study guide version of this document.

[a] `http://tinyurl.com/opdfafk/pub`

Depending on your familiarity with finite difference methods before this course, it might be useful to consult *Finite difference methods for vibration problems* in the course notes[3] since that text describes the fundamentals of the time discretization needed in the present project.

Information on how to structure the software in projects like this (and also use tools for producing reports) is found in *Scientific software engineering with a simple ODE model as example* (in the course notes[4]). Both this document and the one on vibration problems build on *Introduction to computing with finite difference methods*, which was quickly lectured in the beginning of the course.

## 2 The core parts of the project

### 2.1 Mathematical problem

The project addresses the two-dimensional, standard, linear wave equation, with damping,

$$\frac{\partial^2 u}{\partial t^2} + b\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(q(x,y)\frac{\partial u}{\partial x}\right) + \frac{\partial}{\partial y}\left(q(x,y)\frac{\partial u}{\partial y}\right) + f(x,y,t)\,. \tag{1}$$

The associated boundary condition is

$$\frac{\partial u}{\partial n} = 0, \tag{2}$$

in a rectangular spatial domain $\Omega = [0, L_x] \times [0, L_y]$. The initial conditions are

$$u(x,y,0) = I(x,y), \tag{3}$$
$$u_t(x,y,0) = V(x,y)\,. \tag{4}$$

[3] `http://tinyurl.com/opdfafk/pub`
[4] `http://tinyurl.com/opdfafk/pub`

### 2.2 Discretization

Derive the discrete set of equations to be implemented in a program:

- the general scheme for computing $u_{i,j}^{n+1}$ at interior spatial mesh points,
- the modified scheme for the first step,
- the modified scheme at boundary points (first step and subsequent steps), unless you use the interior scheme also at the boundary with extra ghost cells.

### 2.3 Implementation

Implement the numerical method for the PDE problem in a program. You may use `wave2D_u0.py`[5] as a starting point (this program solves the 2D wave equation with constant wave velocity and $u = 0$ on the boundary and is explained in the course notes). You will need to include both scalar (pointwise) computation of the scheme for debugging and reference as well as a vectorized version for speed.

## 3 Verification

### 3.1 Constant solution

Construct a test case with constant solution (not 0 or 1). Make a corresponding nose/pytest test.

Invent five types of possible bugs in the implementation of the mathematical formulas. See how many of them that lead to a wrong non-constant solution.

### 3.2 Cubic solution (optional)

Assume an exact solution on the form $u_{\mathrm{e}}(x, y, t) = X(x)Y(y)T(t)$ where $X$, $Y$, and $T$ are polynomials of degree three or less. Construct $X$ and $Y$ such that the normal derivative vanishes at the four boundaries. Fit a corresponding source term $f(x, y, t)$ in the wave equation.

It would be great if this exact solution also were an exact solution of the discrete equations when $q$ is constant and $b = 0$, which is often the case with lower-order polynomials because the truncation errors[6] involve higher-order derivatives (which vanish for lower-order polynomials).

The exact cubic solution fits the discrete equations at all inner mesh points, *but not on the boundary*. Add a term in the boundary equation for testing such that the exact solution also fulfills the boundary equation. Implement a corresponding nose/pytest test. You will need to introduce a parameter in front of this extra term so you can easily turn the term on and off (it should, of course, only be on for verifying a cubic solution, not in physical applications).

[5] `http://tinyurl.com/nm5587k/wave/wave2D_u0/wave2D_u0.py`
[6] `http://tinyurl.com/opdfafk/pub/trunc/sphinx/index.html`

**Hint.** We outline some ideas in 1D. For constant $q$ and $b = 0$ we have the scheme $[D_tD_tu - qD_xD_x = f]_i^n$ at inner points. Because $[D_xD_xx^3]_i = 6x_i$, $[D_xD_xx^2]_i = 2$, $[D_xD_xx]_i = 0$ all are exact, the suggested $u$ also fits the discrete equation. On the boundary we get a modified scheme, which in operator notation can be written as

$$[D_tD_tu = qD_xD_xu + q\frac{2}{\Delta x}D_{2x}u + f]_i^n, \quad i = 0,$$

and

$$[D_tD_tu = qD_xD_x - q\frac{2}{\Delta x}D_{2x}u + f]_i^n, \quad i = N_x\,.$$

We have the results $[D_{2x}x^3]_i = 3x_i^2 + \Delta x^2$, and $[D_{2x}x^2]_i = 2x_i$, $[D_{2x}x]_i = 1$. Consider now $[D_tD_tu = qD_xD_xu - q\frac{2}{\Delta x}D_{2x}u + f]_i^n$. Inserting $u = X(x)T(t)$ requires the same $f$ as for the PDE, but with an additional term $T(t)2q\Delta x$ because of the $D_{2x}$ operator acting on a cubic polynomial in $x$.

This test requires the $f$ that fits the PDE to be modified on the boundary. A possible implementation is to modify the array of $f$ values at the boundary mesh points directly, or perform tests on the coordinates if a pointwise evaluation of $f$ is requested:

```
def f(x, y, t):
    if isinstance(x, np.ndarray) and isinstance(y, np.ndarray):
        # Array evaluation
        f_a = .... # evaluate the f that fits the PDE
        # Modify boundary values
        f_a[0,:]  = ... # x=0
        f_a[-1,:] = ... # x=Lx
        f_a[:,0]  = ... # y=0
        f_a[:,-1] = ... # x=Ly
    else:
        # Assume pointwise evaluation
        tol = 1E-14  # tolerance for float comparison
        f_v = ... # evaluate the f that fits the PDE
        # Modify boundary values
        if abs(x) < tol:
            f_v = ...   # x=0
        if abs(x-Lx) < tol:
            f_v = ...   # x=Lx
        if abs(y) < tol:
            f_v = ...   # y=0
        if abs(y-Ly) < tol:
            f_v = ...   # y=Ly
```

## 3.3 Exact 1D plug-wave solution in 2D

The program `wave1D_dn_vc.py`[7] has a `pulse` function for simulating the propagation of a plug wave, where $I(x)$ is constant in some region of the domain and zero elsewhere. With unit Courant number, the plug is split into two identical waves, moving in opposite direction, exactly one cell per time step. The discrete solution is then equal to the exact solution.

[7] `http://tinyurl.com/nm5587k/wave/wave1D/wave1D_dn_vc.py`

Set $b = 0$ and $q$ to a constant. Test the 2D program using a one-dimensional plug wave in $x$ direction with $c\Delta t/\Delta x = 1$ (the plug is constant in $y$ direction and hence compatible with the $\partial/\partial y = 0$ boundary condition). Also propagate a one-dimensional plug wave in the $y$ direction with $c\Delta t/\Delta y = 1$. Both test cases are essentially 1D test cases, and the results should as in the 1D case. Implement a corresponding nose test.

### 3.4 Standing, undamped waves

With an exact analytical solution of the PDE we can compute the error and see how the error approaches zero as $\Delta t, \Delta x, \Delta y \to 0$. (See the textbook material[8] for how this is done in a corresponding 1D problem). With no damping ($b = 0$) and constant wave velocity ($c$), our wave equation problem without any source term admits a standing wave solution:

$$u_{\mathrm{e}}(x, y, t) = A\cos(k_x x)\cos(k_y y)\cos(\omega t), \quad k_x = \frac{m_x\pi}{L_x},\ k_y = \frac{m_y\pi}{L_y}, \tag{5}$$

for arbitrary amplitude $A$, arbitrary integers $m_x$ and $m_y$, and a suitable choice of $\omega$. This solution can be used to test the convergence rate of the numerical method.

Compute the true error $e^n_{i,j} = u_{\mathrm{e}}(x_i, y_j, t_n) - u^n_{i,j}$ on a series of meshes. A suitable error norm can be

$$E = ||e^n_{i,j}||_{\ell^\infty} = \max_i \max_j \max_t |e^n_{i,j}|,$$

Introduce a common discretization parameter $h$ such that $\Delta t$, $\Delta x$, and $\Delta y$ are proportional to $h$. This leads to an error model $E = \hat{C}h^r$ for some constant $\hat{C}$ and $r = 2$. Compute a sequence of $r$ values by comparing two consecutive experiments (as shown in the course material) and see if $r$ approaches 2.

### 3.5 Standing, damped waves (optional)

Try to find an analytical solution of damped waves using an ansatz of the type

$$u_{\mathrm{e}}(x, y, t) = (A\cos(\omega t) + B\sin(\omega t))\, e^{-ct}\cos(k_x x)\cos(k_y y m_y y\pi/L_y), \quad k_x = \frac{m_x\pi}{L_x},\ k_y = \frac{m_y\pi}{L_y}\,. \tag{6}$$

That is, find $A$, $B$, $\omega$, and $c$ such that (6) solves the PDE with constant $q$, no source term, and initial condition $u_t(x, y, 0) = 0$ (as for the undamped standing waves). Make a corresponding convergence test.

[8] `http://tinyurl.com/opdfafk/pub/wave/sphinx/._main_wave001.html#manufactured-solution`

**Hint.** The algebra can quickly be quite involved. Getting an overview of the algebra in a 1D version of this problem might be helpful. Start with relating $A$ and $B$ through the initial conditions ($u = A\cos k_x x \cos k_y y$ and $u_t = 0$ as implied by (5)) and eliminate $B$. After having inserted $u$ in the PDE, two equations for $\omega$ and $c$ arise from factoring the sine and cosine terms in time. One equation can be solved for $\omega = \sqrt{k_x^2 q + k_y^2 q - c^2}$, while the other can be solved for $c = b/2$ by inserting the found $\omega$ expression.

### 3.6 Manufactured solution

Choose some $q(x, y) \neq 0$ and find $f(x, y, t)$ such that (wave:app:exer:standing:waves:damped) is a solution to the general 2D wave equation problem with damping and variable wave velocity. Find corresponding $I$ and $V$, and make a convergence test that recovers the expected convergence rate. Make a corresponding nose test.

**Hint.** You may explore `sympy` for automating the analytical work:

```
>>> from sympy import *
>>> x = Symbol('x')
>>> q=x**2
>>> u=sin(x)
>>> r = diff(q*diff(u, x), x)   # Derivative: (q*u_x)_x
>>> simplify(r)
x*(-x*sin(x) + 2*cos(x))
```

## 4 Investigate a physical problem

The purpose of this part is to explore what happens to a wave that enters a medium with different wave velocity. A particular physical interpretation can be wave propagation of a tsunami over a subsea hill. The unknown $u(x, y, t)$ is then the elevation of the ocean surface, and the boundary condition $\partial u/\partial n = 0$ means that the waves are perfectly reflected, because of a steep hill at the shore, or the condition expresses symmetry in the solution. The wave velocity is in this case given by $q = gH(x, y)$, where $g$ is the acceleration of gravity and $H(x, y)$ is the stillwater depth.

It can be wise to do Problem 21[9], because that 1D program, which corresponds to the present 2D problem, allows for much faster experimentation with parameters and effects.

The initial surface is taken as a smooth Gaussian function

$$I(x; I_0, I_a, I_m, I_s) = I_0 + I_a \exp\left(-\left(\frac{x - I_m}{I_s}\right)^2\right), \tag{7}$$

with $I_m = 0$ reflecting the location of the peak of $I(x)$ and $I_s$ being a measure of the width of the function $I(x)$ ($I_s$ is $\sqrt{2}$ times the standard deviation of the familiar normal distribution curve).

[9] `http://tinyurl.com/opdfafk/pub/wave/sphinx/._main_wave010.html#problem-23-earthquake-generated-tsunami-`

Three different bottom shapes can be investigated. A 2D Gaussian hill can be modeled by

$$B(x; B_0, B_a, B_{mx}, B_{my}, B_s, b) = B_0 + B_a \exp\left(-\left(\frac{x - B_{mx}}{B_s}\right)^2 - \left(\frac{y - B_{my}}{bB_s}\right)^2\right), \tag{8}$$

where $b$ is a scaling parameter: $b = 1$ gives a circular Gaussian function with circular contour lines, while $b \neq 1$ gives an elliptic shape with elliptic contour lines.

A less smooth hill is modeled by the "cosine hat" function

$$B(x; B_0, B_a, B_{mx}, B_{my}, B_s) = B_0 + B_a \cos\left(\pi \frac{x - B_{mx}}{2B_s}\right) \cos\left(\pi \frac{y - B_{my}}{2B_s}\right), \tag{9}$$

when $0 \leq \sqrt{x^2 + y^2} \leq B_s$ and $B = B_0$ outside this circle.

A more dramatic hill shape is a box:

$$B(x; B_0, B_a, B_m, B_s, b) = B_0 + B_a \tag{10}$$

for $x$ and $y$ inside a rectangle

$$B_{mx} - B_s \leq x \leq B_{mx} + B_s, \quad B_{my} - bB_s \leq y \leq B_{my} + bB_s,$$

and $B = B_0$ outside this rectangle. The $b$ parameter controls the rectangular shape of the cross section of the box.

Note that the initial condition and the listed bottom shapes are symmetric around the line $y = B_{my}$. We therefore expect the surface elevation also to be symmetric with respect to this line. This means that we can halve the computational domain by working with $[0, L_x] \times [0, B_{my}]$. Along the upper boundary, $y = B_{my}$, we must impose the symmetry condition $\partial\eta/\partial n = 0$. Such a symmetry condition ($-\eta_x = 0$) is also needed at the $x = 0$ boundary because the initial condition has a symmetry here. At the lower boundary $y = 0$ we also set a Neumann condition (which becomes $-\eta_y = 0$). The wave motion is to be simulated until the wave hits the reflecting boundaries where $\partial\eta/\partial n = \eta_x = 0$.

Investigate how different hill shapes, different sizes of the water gap above the hill, and different resolutions $\Delta x = \Delta y = h$ and $\Delta t$ influence the numerical quality of the solution. One anticipates that the less smooth hill shapes will introduce more numerical noise. Presenting the results as movies of the surface elevation is effective.

# 5 Optional additional tasks

## 5.1 Truncation error

Compute the *truncation error* of the scheme at an arbitrary interior mesh point. (It easier to start with $q = \text{const}$ and then generalize to variable $q$).

Suitable background material is the writings on *Truncation error analysis* in the course notes[10].

## 5.2 Harmonic averaging

Harmonic means are often used if the coefficient $q$ is non-smooth or discontinuous. Investigate if harmonic averaging of $q$ works better than the arithmetic averging for the box-shaped subsea hill. The effect might not be big unless the water gap at the top of the hill is small. It can be wise to test the effect of harmonic averaging in 1D first.

**Remark.** With a small gap between the obstruction and the free surface, and with abrupt changes in the bottom shape, the model PDE does not necessarily describe the wave motion in an accurate or qualitatively correct way.

## 5.3 Visualization

Create some fancy 3D visualization of the water waves *and* the subsea hill. Try to make the hill transparent. Suitable tools are

- Mayavi[11]
- Paraview[12]
- OpenDX[13]
- Matplotlib[14]

## 5.4 Open outlet boundary: 1D condition

Implement an open boundary condition at $x = L_x$, $u_t + \sqrt{q}u_x = 0$, as suggested in Problem 21[15]. This condition is only correct in 1D, but might work satisfactorily in 2D if the wave is approximately one-dimensional when it hits the boundary. See how well this condition works in letting the tsunami pass out of the domain. The distance from the subsea hill (which disturbes the wave) and the outlet boundary $x = L_x$ is an important parameter.

[10] `http://hplgit.github.io/num-methods-for-PDEs/doc/pub`
[11] `http://code.enthought.com/projects/mayavi/`
[12] `http://www.paraview.org/`
[13] `http://www.opendx.org/`
[14] `http://matplotlib.org/`
[15] `http://tinyurl.com/opdfafk/pub/wave/sphinx/._main_wave010.html#problem-21-implement-open-boundary-condi`

## 5.5 Open outlet boundary: absorbing layer

Instead of using a condition $u_t + \sqrt{q}u_x = 0$, which is exact only for plane waves propagating in $x$ direction, one can add an artificial domain $[L_x, L_x + \delta] \times [0, L_y]$ where waves are sufficiently damped and absorbed. The goal of an open boundary condition is to avoid waves being reflected back into the domain. Turn on the damping parameter $b$ in $[L_x, L_x + \delta] \times [0, L_y]$, and test if it is wise to vary $b$, say in a linear or exponential fashion to have a smooth transition from $b = 0$ in the physical domain and to some significant (efficient) $b$ value towards the artificial boundary $x = L_x + \delta$.

## 5.6 Open outlet boundary: layer with faked damping

To let waves pass out of the boundary, we can extend the domain and damp the wave as suggested in the section above. However, instead of implementing a physical relevant damping via the term $bu_t$, one can simply multiply the solution by a function that reduces the amplitude of the wave in the extension of the domain. Such a function should be 1 close to the physical boundary and decrease towards 0 at the end of the extened domain. The goal is to let waves pass out of the physical boundary with no reflections back into the physical domain.

## 5.7 Compiled loops

Extend the program with compiled loops using one or more of the following techniques:

- Cython code[16]
- Fortran code[17] interfaced via `f2py`
- C code: interfaced via Cython[18] or f2py[19]
- C/C++ code interfaced via scipy.weave[20]
- C/C++ code interfaced via Instant[21]

Note that Instant comes with FEniCS (`sudo apt-get install fenics` on Ubuntu will install Instant) and it is described in the FEniCS book[22].

[16] `http://tinyurl.com/opdfafk/pub/wave/sphinx/._main_wave006.html#migrating-loops-to-cython`
[17] `http://tinyurl.com/opdfafk/pub/wave/sphinx/._main_wave006.html#migrating-loops-to-fortran`
[18] `http://tinyurl.com/opdfafk/pub/wave/sphinx/._main_wave006.html#migrating-loops-to-c-via-cython`
[19] `http://tinyurl.com/opdfafk/pub/wave/sphinx/._main_wave006.html#migrating-loops-to-c-via-f2py`
[20] `http://docs.scipy.org/doc/scipy/reference/tutorial/weave.html`
[21] `https://launchpad.net/instant`
[22] `https://launchpad.net/fenics-book`

## 5.8 Parallel computing

Make a parallel version of the program using one (or more) of the following approaches:

- Automatic OpenMP code in migrated Cython loops using `cython.parallel`[23]
- OpenMP in migrated C or Fortran loops
- Automatic parallelization via NumbaPro[24]
- MPI in migrated C or Fortran loops
- mpi4py MPI programming from Python (distribute vectorized code)

[23] `http://docs.cython.org/src/userguide/parallelism.html`
[24] `http://docs.continuum.io/numbapro/`